

This Page Is Inserted by IFW Operations
and is not a part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

**As rescanning documents *will not* correct images,
please do not report the images to the
Image Problem Mailbox.**

Authentication for Distributed Systems

Thomas Y.C. Woo and Simon S. Lam
University of Texas at Austin

A distributed system is susceptible to a variety of security threats mounted by intruders. We describe a number of protocols to authenticate users, hosts, and processes.

A distributed system — a collection of hosts interconnected by a network — poses some intricate security problems. A fundamental concern is authentication of local and remote entities in the system. In a distributed system, the hosts communicate by sending and receiving messages over the network. Various resources (like files and printers) distributed among the hosts are shared across the network in the form of network services provided by servers. Individual processes (clients) that desire access to resources direct service requests to the appropriate servers. Aside from such client-server computing, there are many other reasons for having a distributed system. For example, a task can be divided into subtasks that are executed concurrently on different hosts.

A distributed system is susceptible to a variety of threats mounted by intruders as well as legitimate users of the system. Indeed, legitimate users are more powerful adversaries, since they possess internal state information not usually available to an intruder (except after a successful penetration of a host). We identify two general types of threats.

The first type, *host compromise*, refers to the subversion of individual hosts in a system. Various degrees of subversion are possible, ranging from the relatively benign case of corrupting process state information to the extreme case of assuming total control of a host. Host compromise threats can be countered by a combination of hardware techniques (like processor protection modes) and software techniques (like reference monitors). Because these techniques are outside our scope, we refer interested readers to Denning¹ for an overview of computer systems security. Here, we assume that each host implements a reference monitor that can be trusted to properly segregate processes.

The second type, *communication compromise*, includes threats associated with message communications. We subdivide these into

- (T1), eavesdropping of messages transmitted over network links to extract information on private conversations;
- (T2), arbitrary modification, insertion, and deletion of messages transmitted over network links to confound a receiver into accepting fabricated messages; and
- (T3), replay of old messages (a combination of (T1) and (T2)).

(T1) is a passive threat, while (T2) and (T3) are active. A passive threat does not affect the system being threatened, whereas an active threat does. Therefore, passive threats are inherently undetectable by the system and can only be dealt with by using preventive measures. Active threats, on the other hand, are combated by a combination of prevention, detection, and recovery techniques. We will not consider threats of "traffic analysis" and "denial of service" because they are more relevant to the general security of a distributed system than to our restricted setting of authentication.

Some basic security requirements can be formulated. For example, secrecy and integrity are two common requirements for secure communication. Secrecy specifies that a message can be read only by its intended recipients, while integrity specifies that every message is received exactly as it was sent, or a discrepancy is detected.

A strong cryptosystem can provide a high level of assurance for secrecy and integrity (see "Basic cryptography" sidebar). More precisely, an encrypted message provides no information regarding the original message, hence guaranteeing secrecy; and an encrypted message, if tampered with, would not decrypt into a legal message, hence guaranteeing integrity.

Replay of old messages can be countered by using *nonces* or time stamps.^{1,2} A nonce is information that is guaranteed fresh, that is, it has not appeared or been used before. Therefore, a reply that contains some function of a recently sent nonce should be considered timely because the reply could have been generated only after the nonce was sent. Perfect random numbers are good nonce candidates; however, their effectiveness is dependent upon the randomness that is practically achievable. Time stamps are values of a local clock. Their use requires at least some loose synchronization of all local clocks, and hence their effectiveness is also somewhat restricted.

What needs authentication?

In simple terms, authentication is identification plus verification. *Identification* is the process whereby an entity

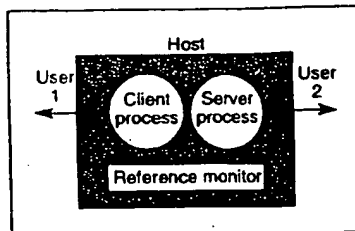


Figure 1. Principals in a distributed system.

claims a certain identity, while *verification* is the process whereby that claim is checked. Thus, the correctness of an authentication relies heavily on the verification procedure employed.

The entities in a distributed system that can be distinctly identified are collectively referred to as *principals*. There are three main types of authentication in a distributed computing system:

- (A1). message content authentication — verifying that the content of a received message is the same as when it was sent;
- (A2). message origin authentication — verifying that the sender of a received message is the same one recorded in the sender field of a message; and
- (A3). general identity authentication — verifying that a principal's identity is as claimed.

(A1) is commonly handled by tagging a key-dependent message authentication code (MAC) onto a message before it is sent. Message integrity can be confirmed upon reception by recomputing the MAC and comparing it with the one attached. (A2) is a subcase of (A3). A successful general identity authentication usually results in a belief held by the authenticating principal (the verifier) that the authenticated principal (the claimant) possesses the claimed identity. Hence, subsequent claimant actions are attributable to the claimed identity; for example, general identity authentication is needed for both authorization and accounting functions. Here, we restrict our attention to general identity authentication.

In an environment where both host and communication compromises can occur, principals must adopt a mutually suspicious attitude. Therefore, mutual authentication, whereby both commu-

nicating principals verify each other's identity, rather than one-way authentication, whereby only one principal verifies the identity of the other principal, is usually required.

In a distributed computing environment, authentication is carried out using a protocol involving message exchanges. We refer to these protocols as authentication protocols.

Most existing systems use only very primitive authentication measures or none at all. For example,

- The prevalent login procedure requires users to enter their passwords in response to a system prompt. Users are then one-way authenticated by verifying the (possibly transformed) password against an internally stored table. However, no mechanism lets users authenticate a system. This design is acceptable only when the system is trustworthy, or the probability of compromise is low.

In a typical client-server interaction, the server — on accepting a client's request — has to trust that (1) the resident host of the client has correctly authenticated the client and (2) the identity supplied in the request actually corresponds to the client. Such trust is valid only if the system's hosts are trustworthy and its communication channels are secure.

These measures are seriously inadequate because the notion of trust in distributed systems is poorly understood. A satisfactory formal explication of trust has yet to be proposed. Second, the proliferation of large-scale distributed systems spanning multiple administrative domains has produced extremely complex trust relationships.

In a distributed computing system, the entities that require identification are hosts, users, and processes.³ They thus constitute the principals involved in an authentication, which we describe (also see Figure 1).

Hosts. These are addressable entities at the network level. A host is distinguished from its underlying supporting hardware. For example, host *H* running on workstation *A* can be moved to workstation *B* by performing the bootstrap sequence for *H* on *B*. A host is usually identified by its name (for example, a domain name) or its network address (for example, an Internet address).

Basic cryptography

A cryptosystem comes with one procedure for encryption and another for decryption. A formal description of a cryptosystem includes specifications for message, key, and ciphertext spaces, and encryption and decryption functions.

There are two broad classes of cryptosystems, symmetric and asymmetric.¹ In the former, encryption and decryption keys are the same and hence must be kept secret. In the latter, the encryption key differs from the decryption key, and the decryption key is kept secret. The encryption key, however, can be made public. Consequently, it is important that no one be able to determine the decryption key from the encryption key. Symmetric and asymmetric cryptosystems are also referred to as shared key and public key cryptosystems, respectively.

Knowledge of the encryption key allows one to encrypt arbitrary messages from the message space, while knowledge of the decryption key allows one to recover a message from its encrypted form. Thus, the encryption and decryption functions satisfy the following relation: M is the message space, $K_e \times K_d$ is the set of encryption/decryption key pairs:

$$\forall m \in M. \forall (k, k') \in K_e \times K_d: \{(m)\}_{k'} = m \quad (C1)$$

where $\{(x)\}_k$ denotes the encryption operation on message x if y is an encryption key and the decryption operation on x if y is a decryption key. (In the case of a symmetric cryptosystem with identical encryption and decryption keys, the operation should be clear from the context.)

Two widely used cryptosystems are the Data Encryption Standard (DES),² a symmetric system, and RSA,³ an asymmetric system. In RSA, encryption-decryption key pairs satisfy the following commutative property⁴:

$$\forall m \in M. \forall (k, k') \in K_e \times K_d: \{(m)\}_{k'} = m \quad (C2)$$

hence yielding a signature capability. That is, suppose k and k' are P 's asymmetric keys, then $\{(m)\}_{k'}$ can be used as P 's signature on m since it could only have been produced by P , the only principal that knows k' . By (C2), P 's signature is verifiable by any principal with knowledge of k , P 's public key. Note that in (C2), the roles of k and k' are reversed; specifically, k' is used as an encryption key while k functions as a decryption key. To avoid confusion with the more typical roles for k and k' as exemplified in (C1), we refer to encryption by k' as a signing operation. In this article, asymmetric cryptosystems are assumed to be commutative.

Since, in practice, symmetric cryptosystems can operate much faster than asymmetric ones, asymmetric cryptosystems are often used only for initialization/control functions, while symmetric cryptosystems can be used for both initializations and actual data transfer.

References

1. G.J. Simmons, "Symmetric and Asymmetric Encryption," *ACM Computing Surveys*, Vol. 11, No. 4, Dec. 1978, pp. 305-330.
2. *Data Encryption Standard*, FIPS Pub. 46, National Bureau of Standards, Washington, D.C., Jan. 1977.
3. R.L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Comm. ACM*, Vol. 21, No. 2, Feb. 1978, pp. 120-126.
4. W. Diffie and M.E. Hellman, "Privacy and Authentication: An Introduction to Cryptography," *Proc. IEEE*, Vol. 67, No. 3, Mar. 1979, pp. 397-427.

whereas a particular host hardware is usually identified by its factory-assigned serial number (for example, a workstation on an Ethernet can be identified by the unique address of its Ethernet adapter board).

Users. These entities are ultimately responsible for all system activities. In other words, users initiate and are accountable for all system activities. Most access-control and accounting functions are based on users. (For completeness, a special user called *root* can be postulated, who is accountable for system-level activities like process scheduling.) Typical users include humans, as well as accounts maintained in the user database. Note that users are considered to be outside the system boundary.

Processes. The system creates processes within the system boundary to represent users. A process requests and

consumes resources on behalf of its unique associated user. Processes fall into two classes: client and server. Client processes are consumers who obtain services from server processes, who are service providers. A particular process can act as both a client and a server. For example, print servers are usually created by (and hence associated with) the user *root* and act as servers for printing requests by other processes. However, they act as clients when requesting files from file servers.

Authentication exchanges

We identify the following major types of authentication exchanges in a distributed system.

Host-host. Host-level activities often

require cooperation between hosts. For example, individual hosts exchange link information for updating their internal topology maps. In remote bootstrapping, a host, upon reinitialization, must identify a trustworthy boot server to supply the information (for example, a copy of the operating system) required for correct initialization.

User-host. A user gains access to a distributed system by logging in a host in the system. In an open-access environment where hosts are scattered across unrestricted areas, a host can be arbitrarily compromised, necessitating mutual authentication between the user and host.

Process-process. Two main subclasses exist:

- **Peer-process communication.** Peer processes must be satisfied with each

other's identity before private communication can begin.

• Client-server communication. An access decision concerning a client's request can be made only when the client's identity is affirmed. A client is willing to surrender valuable information to a server only after verifying the server's identity.

As shown later, these two classes of communication authentication relate closely and can be handled by similar protocols.

From now on, we use authentication to refer to general identity authentication.

Paradigms of authentication protocols

Authentication in distributed systems is always carried out with protocols. A protocol is a precisely defined sequence of communication and computation steps. A communication step transfers messages from one principal (sender) to another (receiver), while a computation step updates a principal's internal state. Two distinct states can be identified upon protocol termination, one signifying successful authentication and the other failure.

Although the goal of any authentication is to verify the claimed identity of a principal, specific success and failure states are highly protocol dependent. For example, the success of an authentication during the connection establishment phase of a communication protocol is usually indicated by the distribution of a fresh session key between two mutually authenticated peer processes. On the other hand, in a user login authentication, success usually results in the creation of a login process on behalf of the user.

We present protocols in the following format. A communication step where-by P sends a message M to Q is represented as $P \rightarrow Q: M$, whereas a computation step of P is written as $P: \dots$, where " \dots " is a specification of the computation step. For example, the typical login protocol between host H and user U is given below (f is a one-way function; that is, given y , it is computationally infeasible to find an x such that $f(x) = y$).

```

U → H: U
H → U: "Please enter password"
U → H: p
H      : compute  $y = f(p)$ 
      : retrieve user record ( $U$ ,
      :  $f(\text{password}_U)$ ) from user
      : database
      : if  $y = f(\text{password}_U)$  then
      :   accept; otherwise reject
  
```

We next examine the key ideas that underlie authentication protocol design by presenting several protocol paradigms.

Since authentication protocol paradigms directly use cryptosystems, their basic design principles also follow closely the type of cryptosystem used.

Note that the protocol paradigms illustrate basic design principles only. A realistic protocol is necessarily a refinement of these basic paradigms and addresses weaker environment assumptions, stronger postconditions, or both. Also, a realistic protocol may use both symmetric and asymmetric cryptosystems.

Protocols based upon symmetric cryptosystems. In a symmetric cryptosystem, knowing the shared key lets a principal encrypt and decrypt arbitrary messages. Without such knowledge, a principal cannot obtain the encrypted version of a message or decrypt an encrypted message. Hence, authentication protocols can be designed according to the (SYM) principle: *If a principal can correctly encrypt a message using a key that the verifier believes is known only to a principal with the claimed identity (outside of the verifier), this act constitutes sufficient proof of identity.*

Thus (SYM) embodies the proof-by-knowledge principle for authentication, that is, a principal's knowledge is indirectly demonstrated through encryption (see "Approaches to authentication" sidebar). Using (SYM), we immediately obtain the following basic protocol (k is a symmetric key shared between P and Q).

```

P      : create  $m = \text{"I am P."}$ 
      : compute  $m' = \{m\}_k$ 
P → Q:  $m, m'$ 
Q      : verify  $\{m\}_k \stackrel{?}{=} m'$ 
      : if equal then accept;
      : otherwise reject
  
```

Clearly, the (SYM) design principle is sound only if the underlying crypto-

system is strong (one cannot find the encrypted version of a message without knowing the key) and the key is secret (it is shared only between the real principal and the verifier). Note that this protocol performs only one-way authentication. Mutual authentication can be achieved by reversing the roles of P and Q .

One major weakness of the protocol is its vulnerability to replays. More precisely, an adversary could masquerade as P by recording the message m' and later replaying it to Q . As mentioned, replay attacks can be countered by using nonces or time stamps. We modify the protocol by adding a challenge-and-response step using nonces (n is a nonce).

```

P → Q: "I am P."
Q → P:  $n$ 
P      : compute  $n' = \{n\}_k$ 
P → Q:  $n'$ 
Q      : verify  $\{n\}_k \stackrel{?}{=} n'$ 
      : if equal then accept;
      : otherwise reject
  
```

Replay is foiled by the freshness of n . Thus, even if an eavesdropper has monitored all previous authentication conversations between P and Q , it still could not produce the correct n' . (This also points out the need for the cryptosystem to withstand known plaintext attack. In other words, the cryptosystem must be unbreakable given the knowledge of plaintext-ciphertext pairs.) The challenge-and-response step can be repeated any number of times until the desired level of confidence is reached by Q .

This protocol is impractical as a general large-scale solution because each principal must store in memory the secret key for every other principal it would ever want to authenticate. This presents major initialization (the predistribution of secret keys) and storage problems. Moreover, the compromise of one principal can potentially compromise the entire system. These problems can be significantly reduced by postulating a centralized authentication server A that shares a secret key k_x with every principal X in the system.² The basic authentication protocol then becomes

```

P → Q: "I am P."
Q → P:  $n$ 
P      : compute  $n' = \{n\}_{k_x}$ 
P → Q:  $n'$ 
Q      : compute  $n'' = \{P, n'\}_{k_x}$ 
  
```

$Q \rightarrow A: n''$
 A : recover (P, n') from n'' by
 decrypting with k_Q
 : compute $m = \{[n']_{k_P}\}_{k_Q}$
 $A \rightarrow Q: m$
 Q : verify $[n]_{k_Q} \stackrel{?}{=} m$
 : if equal then accept;
 otherwise reject

Thus Q 's verification step is preceded by A 's key translation step. The protocol correctness now also rests on A 's trustworthiness — that A will properly decrypt using P 's key and reencrypt using Q 's key. The initialization and storage problems are greatly alleviated because now each principal needs to keep only one key. The risk of compromise is mostly shifted to A , whose security can be guaranteed by various measures, such as encrypting stored keys using a master key and putting A in a physically secure room.

Protocols based upon asymmetric cryptosystems. In an asymmetric cryptosystem, each principal P publishes its public key k_P and keeps secret its private key k_P^{-1} . Thus only P can generate $[m]_{k_P}$ for any message m by signing it using k_P^{-1} . The signed message $[m]_{k_P}$ can be verified by any principal with knowledge of k_P (assuming a commutative asymmetric cryptosystem). The basic design principle is (ASYM): *If a principal can correctly sign a message using the private key of the claimed identity, this act constitutes sufficient proof of identity.*

This (ASYM) principle follows the proof-by-knowledge principle for authentication, in that a principal's knowledge is indirectly demonstrated through its signing capability. Using (ASYM), we obtain a basic protocol as follows (n is a nonce):

$P \rightarrow Q: "I \text{ am } P."$
 $Q \rightarrow P: n$
 P : compute $n' = [n]_{k_P}$
 $P \rightarrow Q: n'$
 Q : verify $n \stackrel{?}{=} [n']_{k_P}$
 : if equal then accept;
 otherwise reject

This protocol depends on the guarantee that $[n]_{k_P}$ cannot be produced without the knowledge of k_P^{-1} and the correctness of k_P , as published by P and kept by Q .

As in protocols that use symmetric keys, the initialization and storage prob-

Approaches to authentication

All authentication procedures involve checking known information about a claimed identity against information acquired from the claimant during the identity-verification procedure. Such checking can be based on the following three approaches.¹

Proof by knowledge. The claimant knows information regarding the claimed identity that can only be known or produced by a principal with that identity. For example, password knowledge is necessary to most login procedures. A proof by knowledge can be conducted by a direct demonstration, like typing in a password, or by an indirect demonstration, such as correctly computing replies to verifier challenges. Direct demonstration is not preferable from a security viewpoint, since a compromised verifier can record the submitted knowledge and later impersonate the claimant by presenting the recorded knowledge. Indirect demonstration can be designed to induce high confidence in the verifier without leaving any clue to how the claimant's replies are computed. For example, Feige, Fiat, and Shamir propose a zero-knowledge protocol for proof of identity. This protocol allows claimant C to prove to verifier V that C knows how to compute replies to challenges without revealing the replies. These protocols are provably secure (under complexity assumptions). However, additional refinements are needed before they can be applied in practical systems.

Proof by possession. The claimant produces an item that can only be possessed by a principal with the claimed identity, for example, an ID badge. The item has to be unforgeable and safely guarded.

Proof by property. The verifier directly measures certain claimant properties with such biometric techniques as fingerprint and retina print. The measured property has to be distinguishing, that is, unique among all possible principals.

Proof by knowledge and possession (and combinations thereof) can be applied to all types of authentication needs in a secure distributed system, while proof by property is generally limited to the authentication of human users by a host equipped with specialized measuring instruments.

References

1. K. Shankar, "The Total Computer Security Problem," *Computer*, Vol. 10, No. 6, June 1977, pp. 50-73.
2. U. Feige, A. Fiat, and A. Shamir, "Zero-Knowledge Proofs of Identity," *Proc. ACM Symp. Theory of Computing*, ACM Press, New York, 1987, pp. 210-217.

lems can be alleviated by postulating a centralized certification authority A that maintains a database of all published public keys. The protocol can then be modified as follows:

$P \rightarrow Q: "I \text{ am } P."$
 $Q \rightarrow P: n$
 P : compute $n' = [n]_{k_P}$
 $P \rightarrow Q: n'$
 $Q \rightarrow A: "I \text{ need } P\text{'s public key}."$
 A : retrieve $c = \{P, k_P\}_{k_A}$ from
 key database
 $A \rightarrow Q: P, c$
 Q : recover (P, k_P) from c by
 decrypting with k_A

: verify $n \stackrel{?}{=} [n']_{k_P}$
 : if equal then accept;
 : otherwise reject

Thus public key certificate c represents a certified statement by A that P 's public key is k_P . Other information such as a specified lifetime and the classification of principal P (for mandatory access control) can also be included in the certificate (such information is omitted here). Each principal in the system need only keep a copy of the public key k_A of A .

In this protocol, A is an example of an on-line certification authority, which

supports interactive queries and is actively involved in authentication exchanges. A certification authority can also operate off line so that a public key certificate is issued to each principal only once. The certificate is kept by the principal and forwarded during an authentication exchange, thus eliminating the need to query *A* interactively. Alternatively, the certificate can be kept in an on-line database that is publicly accessible. Forgery is impossible, since a certificate is signed by the certification authority.

Notion of trust. Correctness of both types of protocol paradigms requires more than the existence of secure communication channels between principals and the appropriate authentication servers (or certification authorities). In fact, such correctness is critically dependent on the capability of the servers (authorities) to faithfully follow the protocols. Each principal bases its judgment on its own observations (messages sent and received) and its trust of the server's judgment.

In some sense, less trust is required of a certification authority than of an authentication server, because all information kept by the authority is public (except for its own private key). Furthermore, a certification authority has no way of masquerading as a principal because a principal's private key is not shared.

Our formal understanding of trust in a distributed system is at best inadequate. In particular, a formal understanding of authentication would require both a formal specification of trust and a rigorous reasoning method where-in trust is a basic element.

Authentication protocol failures

Despite the apparent simplicity of the basic design principles for authentication protocols, designing realistic protocols is notoriously difficult. Several published protocols have exhibited subtle security problems.^{1,2,4}

Several reasons for this difficulty exist. Most realistic cryptosystems satisfy algebraic identities additional to those in (C1) and (C2). These extra properties may generate undesirable effects when combined with protocol logic.

Despite the apparent simplicity of basic design principles for authentication protocols, designing realistic protocols is notoriously difficult.

Second, even assuming that the underlying cryptosystem is perfect, unexpected interaction among the protocol steps can lead to subtle logical flaws. Third, assumptions regarding the environment and the capabilities of an adversary are not explicitly specified, making it extremely difficult to determine when a protocol is applicable and what final states are achieved.

We illustrate the difficulty by showing an authentication protocol proposed by Needham and Schroeder¹ that contains a subtle weakness. Symmetric keys k_p and k_Q are shared between *P* and *A*, and *Q* and *A*, respectively, where *A* is an authentication server. Let *k* be a session key.

- (1) $P \rightarrow A: P, Q, n_p$
- (2) $A \rightarrow P: \{n_p, Q, k, \{k, P\}_{k_p}\}_{k_p}$
- (3) $P \rightarrow Q: \{k, P\}_{k_Q}$
- (4) $Q \rightarrow P: \{n_Q\}_k$
- (5) $P \rightarrow Q: \{n_Q + 1\}_k$

The message $\{k, P\}_{k_Q}$ in step 3 can only be decrypted, and hence understood, by *Q*. Step 4 reflects *Q*'s knowledge of *k*, while step 5 assures *Q* of *P*'s knowledge of *k*; hence the authentication handshake is based entirely on knowledge of *k*. The subtle weakness in the protocol arises from the fact that the message $\{k, P\}_{k_Q}$ sent in step 3 contains no information for *Q* to verify its freshness. (Note that only *P* and *A* know *k* to be fresh.) In fact, this is the first message sent to *Q* about *P*'s intention to establish a secure connection. An adversary who has compromised an old session key k' can impersonate *P* by replaying the recorded message $\{k', P\}_{k_Q}$ in step 3 and subsequently executing steps 4 and 5 using k' .

To avoid protocol failures, formal methods may be employed in the design and verification of authentication protocols. A formal design method should embody the basic design principles as

illustrated in the previous section. In addition, informal reasoning should be formalized within a verification method. Such informal reasoning would include statements like "If you believe that only you and Bob know *k*, then you should believe any message you receive encrypted with *k* was originally sent by Bob," which must be formally specified.

Early attempts at formal verification of security protocols mainly follow an algebraic approach.³ Messages exchanged in a protocol are viewed as terms in an algebra. Various identities involving the encryption and decryption operators (for example, (C1) and (C2)) are taken to be term-rewriting rules. A protocol is secure if it is impossible to derive certain terms (for example, the term containing the key) from the terms obtainable by an adversary. The algebraic approach is limited, since it has been used mainly to deal with one aspect of security, namely secrecy. Recently, logical approaches have been proposed to study authentication protocols.⁴ Most of these logics adopt a modal basis, with belief and knowledge as central notions. The logical approaches appear to be more general than the algebraic ones, but they lack the rigorous foundation of more well-established systems like first-order and temporal logics. A satisfactory semantic model for these systems has not been developed. Much research is needed to obtain sound design methods and to formally understand authentication issues.

Authentication framework

We synthesize basic concepts into an authentication framework that can be incorporated into the design of secure distributed systems. We identify five aspects of secure distributed system design and the associated authentication needs. (This section is not exhaustive in scope because other issues may have to be addressed in an actual distributed system security framework. Also, the presented protocols are not meant to be definitive or optimal.) The five aspects are

- Host initializations. All process executions take place inside hosts. Some

hosts (like workstations) also act as system-entry points by allowing user logins. The overall security of a distributed system is highly dependent on the security of each host. However, in an open network environment, not all hosts can be physically protected. Thus resistance to compromise must be built into a host's software to ensure secure operation. This suggests the importance of host software integrity. Loading a host that employs remote initialization with the correct host software is necessary to its proper functioning. In fact, one way to compromise a public host is to reboot the host with incorrect initialization information. Authentication can be used to implement secure bootstrapping.

- **User logins.** User identity is established at login, and all subsequent user activities are attributed to this identity. All access-control decisions and accounting functions are based on this identity. Correct user identification is thus crucial to the functioning of a secure system. Since any host in an open environment is susceptible to compromise, a user should not engage in any activity with a host without first ascertaining the host's identity. A mutual user-host authentication can achieve the required guarantees.

- **Peer communications.** Distributed systems can distribute a task over multiple hosts to achieve a higher throughput or more balanced utilization than centralized systems. Correctness of such a distributed task depends on whether peer processes participating in the task can correctly identify each other. Authentication can identify friend or foe.

- **Client-server interactions.** The client-server model provides an attractive paradigm for constructing distributed systems. Servers are willing to provide service only to authorized clients, while clients are interested in dealing only with legitimate servers. Authentication can be used to verify a potential consumer-supplier relationship.

- **Interdomain communication.** Most distributed systems are not centrally owned or administered; for example, a campus-wide distributed system often interconnects individually administered departmental subsystems. Identifying principals across subsystems requires additional authentication mechanisms.

In the kind of malicious environments postulated in our threats model, some basic assumptions about the system must

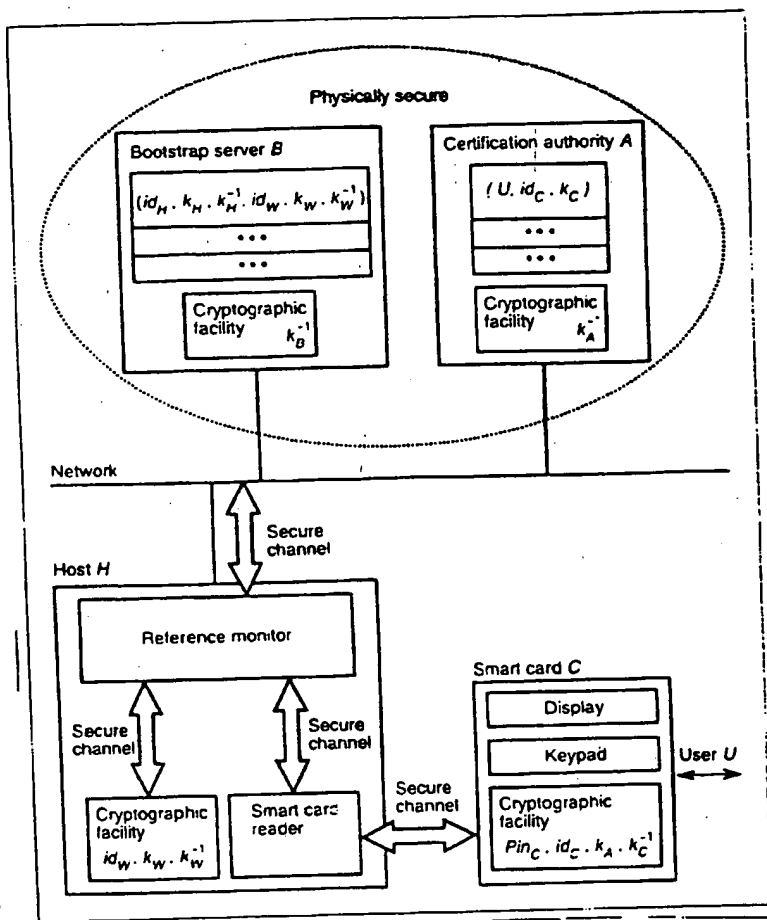


Figure 2. Distributed system configuration.

be satisfied to achieve a reasonable level of security. We offer a set of assumptions (for other possible assumptions, see Abadi et al.⁴ and Linn⁵). Figure 2 also depicts these assumptions.

- Each host hardware W has a unique built-in immutable identity id_W and contains a tamperproof cryptographic facility that encapsulates the public key k_W and the private key k_W^{-1} of W . The cryptographic facility can communicate with the host reference monitor via a secure channel. Each host that supports user logins also has a smart card reader that communicates with the host reference monitor via a secure channel. Lastly, the host reference monitor has a secure channel to the network interface.

- Each legitimate user U is issued a smart card C that has a unique built-in immutable identity id_C . Each smart card C performs encryption and decryption, and encapsulates its private key k_C^{-1} , the public key for the certification authority k_A , and a pin number Pin_C for its legitimate holder. (The pin number is assigned by a card-issuing procedure.) The channel between the smart card and reader is secure. Each smart card has its own display, keypad, and clock.

- A physically secure centralized bootstrap server B maintains a database of host information. More precisely, for each host H , it keeps a record $(id_H, k_H, k_H^{-1}, id_W, k_W, k_W^{-1})$ specifying the unique hardware W that can be initialized as H . All database records can be encrypted under a secret master key for

added security. B has a public key k_B and a private key k_B^{-1} .

• A physically secure centralized certification authority A maintains a database on all principals. More precisely, for each user U , A keeps a record (U, id_U, k_U) , binding U to its smart card C . For each host H , A keeps a record (id_H, id_W) , specifying the hardware W that H can run on. Also, for each server S , A keeps a record of its public key certificate (S, k_S) . The certification authority A has a public key k_A and a private key k_A^{-1} .

Each assumption is achievable with current technology. In particular, the technology of a battery-powered credit-card-sized smart card with a built-in LCD and keypad that can perform specialized computations has steadily progressed. Also, many vendors include specialized cryptographic facilities and smart card readers for hosts as options. The use of a smart card or other forms of computational aid is essential to realizing mutual authentication between a host and a user. Unaided human users simply cannot carry out the intensive computations required by an authentication protocol.

We assume the bootstrap server and certification authority are centralized. Decentralized servers/authorities can be supported by adding authentication between them, as we discuss later. Such authentication can be carried out in a hierarchical manner as suggested in the protocol standard CCITT X.509.

Although we have a certification authority in our framework, we chose to omit an authentication server because we deemed the trust level needed in a certification authority (which distributes public key certificates) to be less than that of an authentication server.

Secure bootstrapping. The following protocol is initiated when host hardware attempts a remote initialization. This could occur after a voluntary shutdown, a system crash, or a malicious attack by an adversary who attempts to penetrate the host. The secure bootstrap protocol allows a reinitialized host

- (1) $W \rightarrow \text{all}$: "boot." $\{id_W, n_W, id_W\}_{k_W^{-1}}$
- (2) B : retrieve record $(id_B, k_B, k_B^{-1}, id_W, k_W, k_W^{-1})$
for W from database
: recover n_W from $\{n_W, id_W\}_{k_W^{-1}}$
by decrypting with k_W^{-1}
: generate a random key k
: compute $m = \{n_W, k_A, k_B, k\}_{k_W}$
- (3) $B \rightarrow W$: m
- (4) W : recover (n_W, k_A, k_B, k) from m
by decrypting with k_W^{-1}
- (5) $W \rightarrow B$: $\{n_W, \text{"ready"}\}_k$
- (6) $B \rightarrow W$: $\{n_W, n_B, id_H, \{k_B^{-1}\}_{k_B}, OS\}_k$
- (7) $W \rightarrow B$: $\{n_B\}_{k_B^{-1}}$
- (8) $B \rightarrow W$: $\{id_H, id_W, k_B, T\}_{k_B^{-1}}$
- (9) H : validate certificate $\{id_H, id_W, k_B, T\}_{k_B^{-1}}$
by encrypting with k_B

Figures 3. Secure bootstrap protocol.

to attain a "safe" state prior to resuming normal operation. In particular, a correctly loaded reference monitor is ready to assume control of the host in this state.

Suppose that the hosts and the bootstrap server B are on the same broadcast network, allowing the message in step 1 of Figure 3 to be received by B . In that protocol, n_W and n_B are nonces, k is a session key, OS is a copy of the operating system, and T is a time stamp.

In step 1, W announces its intention to reboot by broadcasting a boot request. Only B (which has knowledge of k_W^{-1}) can recover the nonce n_W . In step 2, B generates a fresh key k to be used for loading OS . In step 4, W ascertains that m is not a replay by checking the component n_W , since only B could have composed message m . Thus k_A , k_B , and k in m can be safely taken to be the public key of certification authority A , the public key of B , and the session key to be used for loading OS . At this point, B has been authenticated by W .

When the message "ready" encrypted with k is received in step 5, B is certain that the original boot request actually came from W , since only W can decrypt m to retrieve k . Hence, B and W have mutually authenticated each other.

Step 6 is the actual loading of OS and the transferring of host H 's private key k_H^{-1} . OS includes its checksum, which should be recomputed by W to detect any OS tampering in transit. W acknowledges receipt of k_H^{-1} and OS by returning the nonce n_B signed with k_H^{-1} . B verifies that the correct n_B is returned. Then in step 8, a license signed by B affirming the binding of host id_H with public key k_H and hardware id_W is sent to W . After receiving the license, W officially "becomes" H , which retains this license as proof of successful bootstrapping and of its own identity. The time stamp field T within the license denotes its expiration date.

If its secrecy is not required, OS can be transferred unencrypted. However, the checksum of OS must be sent in encrypted form.

User-host authentication. This function occurs when user U walks up to host H and attempts to log in. The authentication requires smart card C . A successful authentication guarantees host H that U is the legitimate holder of C and guarantees user U that H is a "safe" host to use. The host is safe if it holds a valid license (which may have been obtained through secure bootstrapping) and possesses knowledge of the private key k_H^{-1} .

In most systems, the end result of a successful user authentication is the creation of a login process by the host's reference monitor on the user's behalf. The login process is a proxy for the user, and all requests generated by this process are taken as if they are directly made by the user. However, a remote host/server has no way of confirming such proxy status, except to trust the authentication capability and integrity of the local host. Such trust is unacceptable in a potentially malicious environment because a compromised host can simply claim the existence of user login processes to obtain unauthorized services.

This trust requirement can be alleviated if a user explicitly delegates authority to the login host.³⁴ The delegation is carried out by having the

user's smart card sign a login certificate to the login host upon the successful termination of a user-host authentication protocol. The login certificate asserts the host's proxy status with respect to the user and can be presented by the host in future authentication exchanges.

Because forgery can occur, the possession of a login certificate should not be taken as sufficient proof of delegation. The host also must demonstrate the knowledge of a private delegation key k_d^{-1} whose public component k_d is named in the certificate. Also, to reduce the potential impact of a host compromise, the login certificate is given only a finite lifetime by including an expiration time stamp.

We present such a user-host authentication protocol in Figure 4. We assume that the host holds a valid license $\{id_h, id_w, k_h, T\}_{k_c^{-1}}$, as would be the case if the host has executed the secure bootstrap protocol. In the figure, n_c is a nonce, k_d is the public delegation key whose private counterpart k_d^{-1} is kept secret by the host, and T_c is a time stamp denoting the expiration date of the login certificate.

A user inserts his/her smart card into the host's card reader. The card's identity id_c and a nonce n_c are sent through the card reader to the host in step 1. In step 2, H requests user information associated with id_c from certification authority A . Since the license held by H was signed by B and hence is not decipherable by C , a key translation is requested by H in the same step. (Note that these licenses can be cached by H and need not be requested for every user authentication.) After receiving a reply from A in step 4, H knows both the legitimate holder U of the card C and the public key k_c associated with the smart card. Knowledge of U can be used to enforce the local discretionary control to provide service (or not), while k_c is needed to verify the authenticity of C . In step 5, H generates a new delegation key pair (k_d, k_d^{-1}) . H keeps k_d^{-1} private, to be used as proof of a successful delegation from U to H .

- (1) $C \rightarrow H: id_c, n_c$
- (2) $H \rightarrow A: id_c, \{id_h, id_w, k_h, T\}_{k_c^{-1}}$
- (3) A : check time stamp of certificate
: if time stamp expired, abort
- (4) $A \rightarrow H: \{id_h, id_w, k_h, T\}_{k_c^{-1}}, \{U, id_c, k_c\}_{k_d^{-1}}$
- (5) H : generate new delegation key pair (k_d, k_d^{-1})
- (6) $H \rightarrow C: \{id_h, id_w, k_h, T\}_{k_c^{-1}}, \{U, k_d, n_c\}_{k_d^{-1}}$
- (7) $C \rightarrow U: id_h, id_w$
- (8) U : verify if id_h/id_w is the desired host
: if not, abort
- (9) $U \rightarrow C: Pin$
- (10) C : verify $Pin \stackrel{?}{=} Pin_c$
: if not equal, abort
- (11) $C \rightarrow H: \{U, id_h, k_d, T_c\}_{k_d^{-1}}$
- (12) H : verify correct delegation by decrypting the login certificate $\{U, id_h, k_d, T_c\}_{k_d^{-1}}$ with k_c

Figure 4. User-host authentication protocol.

In step 6, H returns the nonce n_c with the public delegation key k_d and a copy of its license to C . In step 7, C retrieves (id_h, id_w) , the identity of H , from the license by decrypting it with k_d . A check ensures that the time stamp in the license has not expired. The identity (id_h, id_w) is then displayed on the card's own screen. To proceed, the user enters the assigned pin number on the card's keypad. In step 10, the pin number is compared with the one stored in the card. If they are equal, C signs a login certificate binding the user U with the host id_h and the public delegation key k_d . This is sent to H in step 11. The host (and others) can verify the validity of the login certificate using k_c , the card's public key.

When user U logs out, the host erases its copy of the private delegation key k_d^{-1} to void the delegation from U . If H is compromised after the delegation, the effect of the login certificate is limited by its lifetime, T_c .

Peer-peer authentication. This type of mutual authentication and cryptographic parameters negotiation is performed in the connection-establishment phase of a secure connection-oriented protocol.

The protocol in Figure 5 mutually authenticates peers P and Q , and estab-

lishes a new session key for their future communication (n_p and n_q are nonces, while k is a fresh session key).

In step 1, P informs A of its intention to establish a secure connection with Q . In step 2, A returns to P a copy of Q 's public key certificate. In step 3, P informs Q of its desire to communicate, and sends a nonce n_p . In step 4, Q asks A for P 's public key certificate and requests a session key at the same time. In order for Q to subsequently prove to P that the session key k is actually from A (not Q 's own creation), A sends a signed statement containing the key k , n_p , and Q 's name. This basically says that k is a key generated by A on behalf of Q 's request identified by n_p . The binding of k and n_p assures P that k is fresh. In step 6, A 's signed copy of (n_p, k, Q) is relayed to P together with a nonce n_q generated by Q . P 's knowledge of the new session key k is indicated to Q by the receipt of n_q in step 7.

Client-server authentication. Since both clients and servers are implemented as processes, the basic protocol for peer-peer authentication can be applied here as well. However, several issues peculiar to client-server interactions need to be addressed.

In a general-purpose distributed computing environment, new services (hence servers) are made available dynamically. Thus, instead of informing clients of every service available, most implemen-

- (1) $P \rightarrow A: P, Q$
- (2) $A \rightarrow P: \{Q, k_Q\}_{k_A^{-1}}$
- (3) $P \rightarrow Q: \{n_p, P\}_{k_Q}$
- (4) $Q \rightarrow A: \{Q, P, (n_p)\}_{k_A}$
- (5) $A \rightarrow Q: \{P, k_p\}_{k_A^{-1}}, \{[n_p, k, Q]_{k_A}, n_q\}_{k_Q}$
- (6) $Q \rightarrow P: \{[n_p, k, Q]_{k_A}, n_q\}_{k_Q}$
- (7) $P \rightarrow Q: \{n_q\}_k$

Figure 5. Peer-peer authentication protocol.

tations use a service broker to keep track of and direct clients to appropriate providers. A client first contacts the service broker by using a purchase protocol that performs the necessary mutual authentication prior to the granting of a ticket. The client later uses the ticket to redeem services from the actual server by using a redemption protocol.

Authentication performed by the purchase protocol proceeds the same way as the protocol for peer-to-peer authentication, while in the redemption protocol authentication is based upon possession of a ticket and knowledge of some information recorded in the ticket. Such a ticket contains the names of the client and server, a key, and a time stamp to indicate lifetime (similar to a login certificate). A ticket can be used only between the specified client and server. A prime example of this approach is the Kerberos authentication system, which we later discuss.

Another special issue of client-server authentication is proxy authentication.⁷ To satisfy a client's request, a server often needs to access other servers on behalf of the client. For example, a database server, upon accepting a query from a client, may need to access the file server to retrieve certain information on the client's behalf. A straightforward solution would require the file server to directly authenticate the client. However, this may not be feasible. In a long chain of service requests, the client may not be aware of a request made by a server in the chain and hence may not be in a position to perform the required authentication. An alternative is to extend the concept of delegation previously used in user-host authentication.⁷ A client can forward a signed delegation certificate affirming the delegation of its rights to a server along with its service request. The server is allowed to delegate to another server by signing its own delegation certificate as well as by relaying the client's certificate. In general, for a service request involving a sequence of servers, delegation can be propagated to the final server through intermediate ones, forming a delegation chain.

```

U → H      : U
H → Kerberos : U, TGS
Kerberos    : retrieve  $k_U$  and  $k_{TGS}$  from database
              : generate a new session key  $k$ 
              : create ticket-granting ticket
                 $tick_{TGS} = \{U, TGS, k, T, L\}_{k_{TGS}}$ 
Kerberos → H :  $\{TGS, k, T, L, tick_{TGS}\}_{k_U}$ 
H → U       : "Password?"
U → H       : passwd
H            : compute  $t = f(passwd)$ 
              : recover  $k, tick_{TGS}$  by decrypting
                 $\{TGS, k, T, L, tick_{TGS}\}_{k_U}$  with  $t$ 
              : if decryption fails, abort login
              : otherwise retain  $tick_{TGS}$  and  $k$ 
              : erase  $passwd$  from memory

```

Figure 6. Kerberos credential-initialization protocol.

Various refinements can extend the scheme. A form of restricted delegation can be carried out by explicitly specifying a set of rights and/or objects in a delegation certificate.

Interdomain authentication. We have assumed a centralized certification authority trusted by all principals. However, a realistic distributed system is often composed of subsystems independently administered by different authorities. We use the term *domain* to refer to such a subsystem. Each domain D maintains its own certification authority A_D that has jurisdiction over all principals within the domain. *Intradomain* authentication refers to an exchange between two principals belonging to the same domain, whereas *interdomain* authentication refers to an exchange that involves two principals belonging to different domains.

Using the previously described protocols, A_D is sufficient for all intradomain authentications for each domain D . However, a certification authority has no way of verifying a request from a remote principal, even if the request is certified by a remote certification authority. Hence, additional mechanisms are required for interdomain authentication.

To allow this type of authentication, two issues need to be addressed: naming and trust. Naming ensures that principals are uniquely identifiable across domains, so that each authentication request can be attributed to a unique principal. A global naming system span-

ning all domains can be used to provide globally unique names to principals. A good example of this is the Domain Name System used in Internet.

Trust refers to the willingness of a local certification authority to accept a certification made by a remote authority regarding a remote principal. Such trust relationships must be explicitly established between domains, which can be achieved by

- sharing an interdomain key between certification authorities that are willing to trust each other,
- installing the public keys of all trusted remote authorities in a local certification authority's database, and
- introducing an interdomain authority for authenticating domain-level authorities.

A hierarchical organization corresponding to that of the naming system can generally be imposed on the certification authorities. In this case, an authentication exchange between two principals P and Q involves multiple certification authorities on a path in the hierarchical organization between P and Q .⁸ The path is referred to as a certification path.

Case studies

We study two authentication services: Kerberos and SPX. Both address client-server authentication needs. Their services are generally available to an application program through a programming interface. While Kerberos uses a symmetric cryptosystem, SPX uses an asymmetric cryptosystem as well.

Kerberos. This system was designed for Project Athena at the Massachusetts Institute of Technology.⁹ The project goal is to create an educational computing environment based on high-performance workstations, high-speed networking, and servers of various types. Researchers envisioned a large-scale (10,000 workstations to 1,000 servers) open-network computing environment in which individual workstations can be

privately owned and operated. Therefore, a workstation cannot be trusted to identify its users correctly to network services. Kerberos is not a complete authentication framework required for secure distributed computing in general; it only addresses issues of client-server interactions.

We limit our discussion to the Kerberos authentication protocols and omit various administrative issues.

The design is based on using a symmetric cryptosystem with trusted third-party authentication servers. It is a refinement of ideas presented in Needham and Schroeder.² The basic components include Kerberos authentication and ticket-granting servers (TGSs). A database contains information on each principal. It stores a copy of each principal's key that is shared with Kerberos. For a user principal U , its shared key k_U is computed from its password $password_U$; specifically, $k_U = f(password_U)$ for some one-way function f . Kerberos servers and TGSs read the database in the course of authentication.

Kerberos uses two main protocols. The credential-initialization protocol authenticates user logins and installs initial tickets at the login host. A client uses the client-server authentication protocol to request services from a server.

The credential-initialization protocol uses Kerberos servers. Let U be a user who attempts to log into host H , and f be the one-way function for computing k_U from U 's password. The protocol is specified in Figure 6 (here, Kerberos refers to the server):

If $password$ is not the valid password of U , k_U would not be identical to k_U , and decryption in the last step would fail. (In practice, f may not be one-to-one. It suffices to require that given two distinct elements x and y , the probability of $f(x)$ being equal to $f(y)$ is negligible.) Upon successful authentication, the host obtains a new session key k and a copy of the ticket-granting ticket

$$tick_{TGS} = \{U, TGS, k, T, L\}_{k_{TGS}}$$

where T is a time stamp and L is the ticket's lifetime. The ticket-granting ticket is used to request server tickets from a TGS; note that $tick_{TGS}$ is encrypted with k_{TGS} , the shared key between TGS and Kerberos.

Because a ticket is susceptible to interception or copying, it does not con-

- (1) $C \rightarrow TGS: S, tick_{TGS}, \{C, T_1\}_k$
- (2) TGS : recover k from $tick_{TGS}$ by decrypting with k_{TGS}
: recover T_1 from $\{C, T_1\}_k$ by decrypting with k
: check timeliness of T_1 with respect to local clock
: create server ticket $tick_s = \{C, S, K', T', L'\}_k$
- (3) $TGS \rightarrow C: \{S, K', T', L', tick_s\}_k$
- (4) C : recover $K', tick_s$ by decrypting with k
- (5) $C \rightarrow S: tick_s, \{C, T_2\}_K$
- (6) S : recover K' from $tick_s$ by decrypting with k_s
: recover T_2 from $\{C, T_2\}_K$ by decrypting with K'
: check timeliness of T_2 with respect to local clock
- (7) $S \rightarrow C: \{T_2 + 1\}_K$

Figure 7. Kerberos client-server authentication protocol.

stitute sufficient proof of identity. Therefore, a principal presenting a ticket must also demonstrate knowledge of the session key k named in the ticket. An authenticator (to be described) provides the demonstration. Figure 7 shows the protocol for client C to request network service from server S (T_1 and T_2 are time stamps).

In step 1, client C presents its ticket-granting ticket $tick_{TGS}$ to TGS to request a ticket for server S . (Note that each client process associates with the unique user who created the process. It inherits the user ID and the ticket-granting ticket issued to the user during login.) C 's knowledge of k is demonstrated using the authenticator $\{C, T_1\}_k$. In step 2, TGS decrypts $tick_{TGS}$, recovers k , and uses it to verify the authenticator. If both step 2 decryptions are successful and T_1 is timely, TGS creates a ticket $tick_s$ for server S and returns it to C . Holding $tick_s$, C repeats the authentication sequence with S . Thus, in step 5, C presents S with $tick_s$ and a new authenticator. In step 6, S performs verifications similar to those performed by TGS in step 2. Finally, step 7 assures C of the server's identity. Note that this protocol requires "loosely synchronized" local clocks for the verification of time stamps.

Kerberos can also be used for authentication across administrative or organizational domains. Each domain is called a realm. Each user belongs to a realm identified by a field in the user's ID. Services registered in a realm will accept only tickets issued by an

authentication server for that realm.

An interrealm key supports cross-realm authentication. The TGS of one realm can be registered as a principal in another by using the shared interrealm key. A user can thus obtain a ticket-granting ticket for contacting a remote TGS from its local TGS. When the ticket-granting ticket is presented to the remote TGS, it can be decrypted by the remote TGS, which uses the appropriate interrealm key to ascertain that the ticket was issued by the user's local TGS. An authentication path spanning multiple intermediate realms is possible.

Kerberos is an evolving system on its fifth version. Bellovin and Merritt¹⁰ discuss limitations of previous versions, some of which have been remedied.

SPX. This authentication service is also intended for open-network environments.¹¹ SPX is a major component of the Digital Distributed System Security Architecture⁹ and its functionalities resemble those of Kerberos. It has credential-initialization and client-server authentication protocols. In addition, it has an enrollment protocol that registers new principals. We focus on the first two protocols and omit the last, along with most other administrative issues.

SPX has a Login Enrollment Agent Facility (LEAF) and a Certificate Distribution Center (CDC) that corresponds to Kerberos servers and TGSs. LEAF, similar to a Kerberos server, is used in the credential-initialization pro-

- (1) $U \rightarrow H$: U, passwd
- (2) $H \rightarrow \text{LEAF}$: $U, \{T, n, h_1(\text{passwd})\}_{k_{\text{LEAF}}}$
- (3) $\text{LEAF} \rightarrow \text{CDC}$: U
- (4) $\text{CDC} \rightarrow \text{LEAF}$: $\{[k_U^{-1}]_{h_2(\text{passwd}_U)}, h_1(\text{passwd}_U)\}_k, [k]_{k_{\text{LEAF}}}$
- (5) LEAF : recover k by decrypting with k_{LEAF}^{-1}
: recover $[k_U^{-1}]_{h_2(\text{passwd}_U)}$ and $h_1(\text{passwd}_U)$ by
decrypting with k
: verify $h_1(\text{passwd}) \stackrel{?}{=} h_1(\text{passwd}_U)$
: if not equal, abort
- (6) $\text{LEAF} \rightarrow H$: $\{[k_U^{-1}]_{h_2(\text{passwd}_U)}\}_n$
- (7) H : recover k_U^{-1} by decrypting first with n and then with
 $h_2(\text{passwd})$
: generate (RSA) delegation key pair (k_d, k_d^{-1})
: create ticket $\text{tick}_U = \{L, U, k_d\}_{k_U^{-1}}$
- (8) $H \rightarrow \text{CDC}$: U
- (9) $\text{CDC} \rightarrow H$: $\{A, k_A\}_{k_U^{-1}}$

Figure 8. STX credential-initialization protocol.

tol. CDC is an on-line depository consisting of public key certificates (for principals and certification authorities) and the encrypted private keys of principals. Note that CDC need not be trustworthy because everything stored in it is encrypted and can be verified independently by principals.

SPX also contains hierarchically organized certification authorities (CAs), which operate off-line and are selectively trusted by principals. Their function is to issue public key certificates (binding names and public keys of principals). Global trust is not needed in SPX. Each CA typically has jurisdiction over just one subset of all principals, while each principal P trusts only a subset of all CAs, referred to as the trusted authorities of P . Systems scalability is greatly enhanced by the absence of global trust and on-line trusted components.

The credential-initialization protocol is performed when a user logs in (see Figure 8). It installs a ticket and a set of trusted-authority certificates for the user upon successful login. In the protocol, U is a user who attempts to log in to host H ; passwd is the password entered by U ; T is a time stamp; L is the lifetime of a ticket; n is a nonce; h_1 and h_2 are publicly known one-way functions; k is a (DES) session key; $k_U, k_{\text{LEAF}}, k_A$ are respectively the public keys of U , the LEAF server, and a trusted authority A of U ; and

k_U^{-1} and k_{LEAF}^{-1} are respectively the private keys of U and LEAF.

In step 1, user U enters its ID and password. In step 2, H applies the one-way function h_1 to the password U entered and sends the result, along with a time stamp T and a nonce n , in a message to LEAF. Upon receiving the message from H , LEAF forwards a request to CDC for U 's private key. This key is stored as a record $([k_U^{-1}]_{h_2(\text{passwd}_U)}, h_1(\text{passwd}_U))$ in CDC. Note that a compromise of CDC would not reveal these private keys. In step 4, CDC sends the requested private-key record to LEAF using a temporary session key k . In step 5, LEAF recovers both $[k_U^{-1}]_{h_2(\text{passwd}_U)}$ and $h_1(\text{passwd}_U)$ from CDC's reply. LEAF then verifies passwd by checking $h_1(\text{passwd})$ against $h_1(\text{passwd}_U)$. If they are not equal, the login session is aborted and the abortion logged. Because $h_1(\text{passwd}_U)$ is not revealed to any principal except LEAF, password-guessing attacks would require contacting LEAF for each guess or compromising LEAF's private key.

Having determined the password to be valid, LEAF sends the first part of the private-key record encrypted by n to H in step 6. (The nonce n sent in step 2 is used as a symmetric key for encryption.) In step 7, H recovers k_U^{-1} by decrypting the reply from LEAF first with n and then with $h_2(\text{passwd})$. H then gen-

erates a pair of delegation keys and creates a ticket tick_U . In step 8, H requests the public key certificate for a trusted authority of U from CDC. CDC replies with the certificate in step 9. In fact, multiple certificates can be returned in step 9 if U trusts more than one CA. These trusted authorities' certificates were previously deposited in the CDC by U using the enrollment protocol.

The authentication-exchange protocol between a client C and a server S follows. To simplify the protocol specification so that a single public key certificate is sent in step 2 and in step 5, we made the following assumption: Let C 's public key certificate be signed by A_C where A_C denotes a trusted authority of S . Similarly, let S 's public key certificate be signed by A_S where A_S denotes a trusted authority of C . Below, T is a time stamp, and k is a (DES) session key:

- (1) $C \rightarrow \text{CDC}$: S
- (2) $\text{CDC} \rightarrow C$: $\{S, k_A\}_{k_C^{-1}}$
- (3) $C \rightarrow S$: $T, [k]_{k_A}, \text{tick}_C, [k_d^{-1}]_{k_C}$
- (4) $S \rightarrow \text{CDC}$: C
- (5) $\text{CDC} \rightarrow S$: $\{C, k_C\}_{k_A^{-1}}$
- (6) S : validate tick_C by
encrypting with k_C
- (7) $S \rightarrow C$: $[T+1]_k$

In step 1, C requests S 's public key certificate from CDC. In step 2, CDC returns the requested certificate. C can verify the public key certificate by decrypting it with k_A , which is the public key of A_S obtained by C when it executed the credential-initialization protocol. In step 3, tick_C and the private delegation key k_d^{-1} generated in step 7 of the credential-initialization protocol, as well as a new session key k , are sent to S . Only S can recover k from $[k]_{k_A}$ and subsequently decrypt $[k_d^{-1}]_{k_C}$ to recover k_d^{-1} . Possession of tick_C and knowledge of the private delegation key constitute sufficient proof of delegation from C to S . However, if such delegation from C to S is not needed,

$\{[k]_{k_A}\}_{k_C^{-1}}$

is sent in step 3 instead of $[k_d^{-1}]_{k_C}$; this acts as an authenticator for proving C 's knowledge of k_d^{-1} without revealing it. In steps 4 and 5, S requests the public key certificate of C , which is used to verify tick_C in step 6. In step 7, S returns $[T+1]_k$ to C to complete mu-

tual authentication between C and S.

Since SPX is a relatively recent proposal, its security properties have not been studied extensively. Such study would be necessary before it could be generally adopted.

Although SPX offers services similar to those of Kerberos, its elimination of on-line trusted authentication servers and the extensive use of hierarchical trust relationships are intended to make SPX scalable for very large distributed systems.

With the growth in scale of distributed systems, security has become a major concern — and a limiting factor — in their design. Security has been strongly advocated as one of the major design constraints in both the Athena project at the Massachusetts Institute of Technology and the Andrew project at Carnegie Mellon University. Most existing distributed systems, however, do not have a well-defined security framework and use authentication only for their most critical applications, if at all.

Most of the protocols we present are practically feasible, and their adoption and use should be just a matter of need.

The complexity of understanding and managing the security of a distributed system requires a formal approach that allows precise specifications of both the system's architecture and its protocols. Lam, Shankar, and Woo¹² have proposed a basis for developing such an approach. ■

Acknowledgments

This work was partly supported by a grant from the Texas Advanced Research Program and by the National Science Foundation Grant NCR-9004464. We thank Clifford Neuman of the University of Washington and John Kohl of the Massachusetts Institute of Technology for reviewing the section on Kerberos, and Joseph Tardo and Kannan Alagappan of Digital Equipment Corp. for reviewing the section on SPX. We are also grateful to the anonymous referees for their constructive comments.

References

1. D.E. Denning, *Cryptography and Data Security*, Addison-Wesley Publishing Co., Reading, Mass.: 1982.
2. R.M. Needham and M.D. Schroeder, "Using Encryption for Authentication in Large Networks of Computers," *Comm. ACM*, Vol. 21, No. 12, Dec. 1978, pp. 993-999.
3. J. Linn, "Practical Authentication for Distributed Computing," *Proc. IEEE Symp. Research in Security and Privacy*, IEEE CS Press, Los Alamitos, Calif., Order No. 2060, 1990, pp. 31-40.
4. M. Burrows, M. Abadi, and R. M. Needham, "A Logic of Authentication," *ACM Trans. Computer Systems*, Vol. 8, No. 1, Feb. 1990, pp. 18-36.
5. D. Dolev and A.C. Yao, "On the Security of Public Key Protocols," *IEEE Trans. Information Theory*, Vol. IT-30, No. 2, Mar. 1983, pp. 198-208.
6. M. Abadi et al., "Authentication and Delegation with Smart Cards," Tech. Report 67, Systems Research Center, Dig-



CALL FOR PAPERS

First International Conference
on

Computer Communications and Networks (IC³N)

June 8 - 10, 1992
San Diego, California



Sponsor: The International Society for Mini and Microcomputers

The first IC³N will be devoted to all aspects of computer communications and networks. Authors are invited to submit all work which will contribute to the state-of-the-art. Topics to be covered include, but are not limited to:

ISDN
FDDI
SONET
ATM Switching
Broadband ISDN

Local Area Networks
Wide Area Networks
High Speed Networks
Interconnection Networks
Metropolitan Area Networks

Satellite Communications
Communication Protocols
Network Management
Network Design and Planning
Network Modeling and Analysis

Four copies of the double-spaced manuscript must be received by one of the Program Co-Chairs no later than February 28, 1992. The papers will be reviewed and notification will be mailed by March 31, 1992. The final camera-ready manuscripts are due by April 27, 1992.

General Chair: Dr. Victor O. K. Li, Department of Electrical Engineering, University of Southern California, Los Angeles, CA 90089-2565, (213) 740-4665; (FAX) (213) 740-7290, vli@irving.usc.edu; Program Co-Chairs: Kijoon Chae, (410) 267-3032 and E. K. Park, (410) 267-3037, Computer Science Department, United States Naval Academy, Annapolis, MD 21402, (FAX) (410) 267-4883, chae@usna.navy.mil (t Dr. Chae), eun@usna.navy.mil (to Dr. Park)

ital Equipment Corp., Palo Alto, Calif., Oct. 1990.

Los Alamitos, Calif., Order No. 2168, 1991, pp. 232-244.

7. M. Gasser and E. McDermott, "An Architecture for Practical Delegation in a Distributed System," *Proc. IEEE Symp. Research in Security and Privacy*, IEEE CS Press, Los Alamitos, Calif., Order No. 2060, 1990, pp. 20-30.

8. M. Gasser et al., "The Digital Distributed System Security Architecture," *Proc. Nat'l Computer Security Conf.*, Nat'l Institute of Standards and Technology, 1989, pp. 305-319.

9. J.G. Steiner, C. Neuman, and J.I. Schiller, "Kerberos: An Authentication Service for Open Network Systems," *Proc. Winter Usenix Conf.*, Usenix Assoc., Berkeley, Calif., 1988, pp. 191-202.

10. S.M. Bellovin and M. Merritt, "Limitations of the Kerberos Authentication System," *Proc. Winter Usenix Conf.*, Usenix Assoc., Berkeley, Calif., 1991, pp. 253-267.

11. J.J. Tardo and K. Alagappan, "SPX: Global Authentication Using Public Key Certificates," *Proc. IEEE Symp. Research in Security and Privacy*, IEEE CS Press,

12. S.S. Lam, A.U. Shankar, and T.Y.C. Woo, "Applying a Theory of Modules and Interfaces to Security Verifications," *Proc. IEEE Symp. Research in Security and Privacy*, IEEE CS Press, Los Alamitos, Calif., Order No. 2168, 1991, pp. 136-154.



Simon S. Lam is a professor of computer sciences at the University of Texas at Austin, where he holds an endowed professorship. His research interests are in computer networks, communication protocols, performance models, formal verification methods, and network security. He serves on the editorial boards of *Performance Evaluation* and *IEEE Transactions on Software Engineering*.

Lam received the BSEE degree (with Distinction) from Washington State University in 1969, and the MS and PhD degrees in engineering from the University of California at Los Angeles in 1970 and 1974. He organized and was program chair of the first ACM SIGComm Symposium on Communications Architectures and Protocols in 1983. He is an IEEE fellow and corecipient of the 1975 Leonard G. Abraham Prize Paper Award from the IEEE Communications Society. He is a member of the Computer Society and ACM.



Thomas Y.C. Woo is a PhD student in the Department of Computer Sciences at the University of Texas at Austin. His research interests include computer networking and distributed systems.

Woo received a BS (First-Class Honors) in computer science from the University of Hong Kong and an MS in computer science from the University of Texas at Austin.

Readers may contact Simon S. Lam at the Department of Computer Sciences, University of Texas at Austin, Austin, TX 78712-1188.

SPONSORED BY:

IEEE COMMUNICATIONS SOCIETY

IEEE ELECTRONIC PACKAGING SOCIETY

OPTICAL FIBER COMMUNICATIONS SOCIETY

OPTICAL FIBER COMMUNICATION

OFC® '92

TELECOM
• LAN
• MAN
• ISDN
• FDDI
• CATV
• VIDEO
• SWITCHING

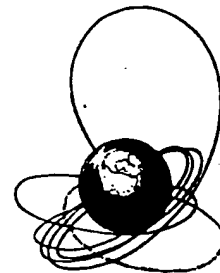
FEBRUARY 2-7, 1992 • SAN JOSE CONVENTION CENTER • SAN JOSE, CALIFORNIA

OFC IS CONSIDERED THE MOST IMPORTANT CONFERENCE ON FIBER-OPTIC SYSTEMS, NETWORKS, COMPONENTS, AND DEVICES FOR COMMUNICATIONS AND RELATED FIELDS.

SEE HOW THE TECHNOLOGIES OF OPTICAL FIBER DEVICES, COMPONENTS, AND SYSTEMS DOMINATE THE WORLD'S COMMUNICATIONS AND RELATED FIELDS.

The OFC Technical conference features over 250 refereed papers in four principal areas: **Fibers, Cables and Glass Technologies; Optoelectronic and Integrated-Optics Devices and Components; System Technologies; and Networks and Switching**

Visit the OFC 350-booth exhibition with over 200 companies displaying the the most current products and services used in components, devices, and fiber-optic systems for the field of communications and related fields.



A New Family of Authentication Protocols

Ross Anderson¹, Francesco Bergadano², Bruno Crispo²,
Jong-Hyeon Lee¹, Charalampos Maniavas¹ and Roger Needham³

¹ Cambridge University Computer Laboratory, England

² Dipartimento di Informatica, Università di Torino, Italy

³ Microsoft Research, Cambridge, England

Abstract. We present a related family of authentication and digital signature protocols based on symmetric cryptographic primitives which perform substantially better than previous constructions. Previously, one-time digital signatures based on hash functions involved hundreds of hash function computations for each signature; we show that given online access to a timestamping service, we can sign messages using only two computations of a hash function. Previously, techniques to sign infinite streams involved one such one-time signature for each message block; we show that in many realistic scenarios a small number of hash function computations is sufficient. Previously, the Diffie Hellman protocol enabled two principals to create a confidentiality key from scratch: we provide an equivalent protocol for integrity, which enables two people who do not share a secret to set up a securely serialised channel into which attackers cannot subsequently intrude. In addition to being of potential use in real applications, our constructions also raise interesting questions about the definition of a digital signature, and the relationship between integrity and authenticity.

Keywords: authentication, non-repudiation, hashing, timestamping

1 Introduction

Most existing cryptographic protocols that provide non-repudiation, whether of origin or receipt, are based on digital signature algorithms such as RSA and DSA. However convenient this may be in some applications, is not necessary: nonrepudiation services have been provided without signatures, and an example is the SWIFT system for international banking transactions that was fielded in the mid 1970's as a replacement for older and less secure telegraphic transfer systems. In SWIFT, pairs of corresponding banks shared MAC keys that were exchanged manually; the messages passed from one bank to another over a private network with multiple independently administered logging facilities (the latest version of SWIFT has digital signatures but since they are apparently applied to the MAC, the logging service is still needed for non-repudiation). However third party logging facilities are expensive and in many applications it

is desired that principals should have the means to generate and store evidence that they might need to press their claims in subsequent disputes.

An early system that did not rely on third party logging was designed by TRW for the NSA in the 1970's to authenticate messages from sensors placed in missile silos to monitor the SALT 2 treaty [1]. It used concatenated encryption: a message would first be encrypted using a Russian device and then with an American one. The keys were made available to the other side (and to third party monitors such as the United Nations) after the messages had been logged by all interested parties. This technique was published in 1983 in the context of the test ban treaty; at that time, it was preferred over RSA because neither of the two superpowers would trust a device built by the other, and asymmetric mechanisms were felt to offer little additional benefit given this constraint [2].

The first published approach to the provision of nonrepudiation without using asymmetric encryption is due to Lamport, who generates signatures by opening commitments that have been made using a one-way hash function [3]. The basic idea here is that the signer chooses two random numbers (representing 0 and 1) for each bit of the message, and publishes their images under a one-way hash function. To sign a message, he reveals the preimages corresponding to the actual 0's and 1's. Despite refinement by Merkle [4, ?], by Even, Goldreich and Micali [6] and by Bleichenbacher and Maurer [7], this technique still requires a lot of hash computations.

In this paper, we show how to construct digital signatures that require only a small number of hash function computations each. This enormous improvement is brought about by making signature interactive; users may either interact with each other or with a time-stamping service. In many applications, interaction with a counterparty or a trusted third party service is a requirement in order to verify the availability of funds, the uniqueness of negotiable instruments or the absence of a key on a certificate revocation list. In this case, it may be possible to dispense with signatures based on number theory.

Possible benefits include the elimination of some patent royalty and export control problems and a measure of insurance against any success that quantum computers have in breaking systems based on number theory. In addition, as there are no long term secrets in our protocols, they might help to overcome intelligence agency concerns that signature keys can easily be abused for encryption. Finally, our constructions raise a number of interesting theoretical points.

2 The Basic Idea

The underlying idea came to us on the 4th November 1996 while discussing how a modern day Guy Fawkes, about to blow up the Houses of Parliament, could arrange publicity for his cause — but without getting caught¹.

¹ for the benefit of readers unfamiliar with British history, Fawkes conspired to blow up King James the first and the Houses of Parliament in 1605; this was an attempt

The naïve approach might be to telephone the newsroom of the 'Times' and say *"I represent the free Jacobin army and we are going to carry out a liberation action tomorrow. Once we have done it, I will call using the code word 'Darnley' to state our demands"*.

2.1 Using hash functions

This is not a particularly secure way of doing things. The message will be passed on to the police, who might remember that the state opening of Parliament is imminent and double the guard. So it would be more prudent to send a hash of the message. Provided that the hash function is truly one-way (technically, pseudorandom), this will not leak information. If Fawkes is now successful, he can reveal the message and find himself possessed of a very credible codeword.

We understand that organisations such as the IRA do in fact share codewords with newspapers and use these to claim credit for their crimes. However, such a protocol is open to abuse by newspaper staff (as well as by other people with access, such as phone company employees and policemen authorised to tap telephone lines). So our next logic step in improving the protocol is to replace the codeword with its hash.

Our protocol is now

- Select a random codeword X
- Form its hash $Y = h(X)$
- Construct a message $M =$ *"We are the free Jacobin army and we are going to blow up the Houses of Parliament tomorrow. The codeword by which we will authenticate ourselves afterwards will be the preimage of Y "*
- Compute $Z = h(M)$ and publish it anonymously
- Blow up the Houses of Parliament
- Reveal M .

This might appear to be only a slightly more technological version of the protocols already used by various liberation groups and criminals. It still suffers from the serious problem that, in the face of a capable motivated opponent, the password is only one-time; once it has been revealed, and is known to the newspapers and the police, any journalist or policeman could in theory masquerade as the rebel leader. Indeed, if Guy Fawkes tries to state his demand by sending the 'Times' the codeword X with a political demand P ('votes for Catholics!'), the police can intercept the message and replace P with the demand P' ('a million guineas for Fawkes!'), thus discrediting Fawkes and his organisation.

to end the persecution of Roman Catholics. Fawkes was caught as a result of a comsec failure (a coded letter from one of the conspirators was intercepted and deciphered). After his public execution, Parliament ordained the 5th of November as a day of thanksgiving for their narrow escape, and it is still celebrated by bonfires and fireworks displays.

2.2 The Guy Fawkes Protocol

Our critical innovation is to introduce a chaining mechanism that lets us bind codewords to messages in a way that provides not just authentication but also nonrepudiation. It also allows the secret codeword to be refreshed, so that the system can be used an arbitrary number of times.

The basic idea is that, at each round of the protocol, we firstly commit to a string consisting of (codeword, message, [hash of next codeword]) by publishing a hash of it. This commitment binds the message to the codeword and its successor. We then reveal the value of this string, proving our knowledge of the codeword and thus authenticating ourselves.

Formally, we define the protocol by induction. Suppose that we have published Z_i followed by the message M_i containing $h(X_i)$, where our secret codeword is currently X_i . We wish to authenticate the message M_{i+1} . We follow the following protocol:

- Select a random codeword X_{i+1}
- Form its hash $h(X_{i+1})$
- Compute $Z_{i+1} = h(M_{i+1}, h(X_{i+1}), X_i)$ and publish it
- Reveal M_{i+1} , $h(X_{i+1})$ and X_i

The first codeword needs to be bootstrapped by some external mechanism; in most applications, this would be a conventional digital signature or an out-of-band authentication, perhaps using a conventional CA. We will give some examples below.

3 Discussion

Hash chains were introduced by Lamport [3] and have been used in one-time password applications such as the S/Key one-time password system [8], as well as several electronic payment protocols [9–11]. There, as here, the effect is to establish secure association at low computational cost.

In S/Key, a user has a series of one-time passwords, each of which is the preimage of its predecessor; the goal is to show that an authorised user is present and protect against passive attacks (though not against active attacks such as session stealing). This chain of events is rooted in a single manual authentication event in which the last element of the hash chain is set as the first password in the system.

In the payment protocols, the goal is to associate a number of electronic coins with a single digital signature that authenticates them all, and thus enable a series of small payments to be made by a customer to a single merchant (such as a phone company) at the cost of a single digital signature or online authentication operation.

In the Guy Fawkes protocol, the objective is to associate a single act of authentication with a stream of future statements rather than a stream of future events. Functionally, the difference is that while the format of all the digital coins is known at the time they are signed, the future statements that we wish to authenticate may not be. So it would not be sufficient to simply use a hash chain (as in S/Key) as a set of one-time passwords for authenticating political statements. As in section 2.1 above, anyone who was tapping the line when the statement and password were sent to the newsroom could alter the statement; and staff in the newsroom could also substitute messages at will.

In other words, the broadcast commitment step has the critical effect of providing nonrepudiation, and gives the Guy Fawkes protocol the same effect as a digital signature. Were the Jacobins permitted to use asymmetric cryptography, then their first message could just as well have read *"We are going to blow up the Houses of Parliament on the 5th November. Future demands will be digitally signed and the public verification key is W."* This could have been encrypted and published, with the key made known after the event.

So we might ask whether there is anything to signature other than secure association. After all, in the conventional model, a digital signature sets up a secure association between something that has been signed at an arbitrary time, and an authentication instance which may have involved showing a passport to a certification authority. Is the Guy Fawkes protocol any different?

4 It may be secure, but it is a signature?

At this point, some might argue that although the Guy Fawkes protocol gives the same effect as a digital signature, it is not actually a signature. However, our protocol satisfies most of the definitions of 'digital signature' offered in the literature to date. We will go through them in turn.

Diffie and Hellman introduced the concept of digital signature in their seminal 'New Directions' paper: *'it must be easy for anyone to recognise the signature as authentic, but impossible for anyone other than the legitimate signer to produce it'* [12]. At the time when this paper was written, the only known way of doing this was using Lamport's one-time signature. The Guy Fawkes protocol improves on Lamport and so, not surprisingly, satisfies this definition; it also satisfies a later definition by Diffie as *'a way of demonstrating to other people that (a message) had come from a particular person'* [13].

Fiat and Shamir refine and extend the definition given by Diffie and Hellman. Authentication is when A can prove to B that she's A, but no-one else can prove to B that he's A; identification is when A can prove to B that she's A, but B cannot prove to anyone else that he's A; and signature is when A can prove to B that she's A, but B can't even prove to himself that he's A [14]. Guy Fawkes satisfies this definition too.

Goldwasser, Micali and Rivest give a more involved description that explicitly mentions a number of algorithms and their properties: a key generation algorithm, a signature algorithm, and a verification algorithm. The signature algorithm produces a signature on being input the message, the key and possibly other information (such as a random input); however, in their definition it produces only a single output [15].

This model therefore excludes the Guy Fawkes protocol. But it also excludes the large class of arbitrated signatures that were already well known and in use by that time (see, for example, [16]) as well as most of the special purpose signature constructions that require interaction, such as undeniable signatures, designated confirmer signatures and oblivious signatures [17].

Naor and Yung refined the approach of Goldwasser, Micali and Rivest, by cutting the complexity theoretic requirement of the construction [18]; it was finally reduced by Rompel to the existence of one-way functions (which is minimal) [19]. However, like Goldwasser, Micali and Rivest, their definitions also fail to deal with signatures that use interaction.

Pfitzmann provides the most thorough study of disparate signature schemes in her thesis [20]. She concludes that the general definition of signature is a process with a number of access points — typically for the signer, the recipient and the court. Time is a necessary component, although logical time (in the sense of a 'global notion of numbered rounds') is sufficient (*op. cit.*, p 54). Special access points can be added for risk bearers such as certification and revocation authorities. This definition clearly admits the Guy Fawkes protocol.

So if it is claimed that the Guy Fawkes protocol is not really a signature, then the onus would be on the objector to show how to deal with the many other kinds of signature that use interaction, as well as the importance of context — the framework of certification and revocation services, legal conventions and so on — to the utility of digital signatures. In most applications, the value of signatures ultimately depends on convention (such as a digital signature law, or a contract between members of an EDI system) and the validation of even conventional digital signatures involves reference to an online or at least near-real-time certificate revocation list.

In passing, we observe that the signatures produced by Guy Fawkes are actually stronger than RSA in the sense that they can be fail-stop at no extra cost: just choose the secrets X_i uniformly at random as bitstrings significantly longer than the hash function's output. That way, an attacker who finds a preimage of a commitment or hashed codeword will with high probability have found a different one from that known to the genuine signer, who will thus be able to exhibit a collision for the hash function.

5 Signing Bidirectional Digital Streams

Hash-based signatures have been condemned as “time-consuming, costly and wasteful” ([16]). Guy Fawkes is much less so than previous schemes; and there are applications for which it might be practical.

Firstly, let us consider the most convincing proposal for a practical application of hash-based signatures. This is the method of Gennaro and Rohatgi for signing digital streams [21]. When signing a stream whose content is not known in advance to the signer (such as a television programme), they divide the stream into blocks; each block contains a one-time public key using the Lamport scheme, and is signed with the one-time private key whose public key was sent in the previous block. The first block is signed using a conventional mechanism such as RSA. In this way, a single conventional signature can be leveraged to sign a whole stream of data ‘on-the-fly’. The authentication thus provided is fast, in that no use is made of asymmetric cryptography once the session is established; but it is bulky, as both a one-time public key and a one-time signature must be added to each block.

The Guy Fawkes mechanism can be adapted readily to this application and can greatly reduce the amount of computation required; it can cope particularly well with bidirectional streams, such as in videoconferencing, although it also works well in applications where a stream is sent to a recipient who merely sends a series of acknowledgements.

Here, our protection goals are that if any bit in the two streams is changed, both communicating parties will detect the problem; and that the authentication mechanisms are as fast as in Gennaro and Rohatgi’s scheme without the message extension (in fact Guy Fawkes is faster). Finally it must provide non-repudiation as well as simple authentication; a third party observing the stream exchange can ascertain the information source and integrity, as opposed to symmetric MACs where the use of shared secrets makes mutual recrimination possible.

In this protocol, Alice and Bob will exchange message streams consisting of sequential blocks which we will call A_0, A_1, A_2, \dots and B_0, B_1, B_2, \dots respectively; each block will be accompanied by authentication information to be described. B_i is sent after A_i but before A_{i+1} .

In addition, Alice will choose a series of passwords X_0, X_1, X_2, \dots ; she will commit to X_i in message A_{i-1} and reveal it in message A_{i+1} . This commitment is called a_i and has the form

$$a_i = h(A_{i+1}, h(X_{i+1}), X_i)$$

Similarly, Bob’s commitments take the form $b_i = h(B_{i+1}, h(Y_{i+1}), Y_i)$. It should be noted that Alice needs a buffer size equal to two blocks; to send message A_i she needs to know A_{i+1} in order to compute the hash value a_i . This will not normally be a problem where, for example, each block is a frame of video.

The first steps of the protocol, which use conventional signatures to bootstrap the process, run as follows:

$$\begin{aligned} A &\rightarrow B : A_0, a_0, h(X_0), \text{sign}_A(A_0, h(X_0)) \\ B &\rightarrow A : B_0, b_0, h(Y_0), \text{sign}_B(B_0, h(X_0)) \\ A &\rightarrow B : h(b_0, X_0) \\ B &\rightarrow A : h(a_0, Y_0) \end{aligned}$$

The authentication of each subsequent block now takes the following form:

$$\begin{aligned} A &\rightarrow B : A_1, a_1, h(X_1), X_0 \\ B &\rightarrow A : B_1, b_1, h(Y_1), Y_0 \\ A &\rightarrow B : h(b_1, X_1) \\ B &\rightarrow A : h(a_1, Y_1) \\ &\dots \end{aligned}$$

Thus in this step, Alice has committed to the password X_2 (since $a_1 = h(A_2, h(X_2), X_1)$) and revealed the password X_0 ; this revelation authenticates A_1 , while the commitment also refreshes the passwords.

The security of this scheme follows inductively. Assuming faithful execution up to step n , and an attacker who tries to masquerade as Bob to Alice, having seen and intercepted the string $B_n, b_n, h(Y_n), Y_{n-1}$. He cannot change B_n as b_{n-1} contains a commitment to it; he cannot change b_n as it contains as an input Y_n , which he doesn't know but which was committed in b_{n-1} ; $h(Y_1)$ was similarly committed in b_{n-1} ; and if he forwards anything other than the correct value of Y_{n-1} then this will fail to verify against b_n and b_{n-1} . Similarly, he cannot in the next message forward anything other than the correct value of $h(a_n, Y_n)$ as he does not know the value of Y_n yet cannot change it, since it was committed at the previous step in n_{n-1} .

As a corollary, we obtain a protocol for authenticating a single digital stream: Alice sends the stream to Bob, and Bob simply sends an ack with a serial number as the text B_n .

There is one final subtlety. If all the passwords are eventually made known, then false content can be cut and pasted at will into a record of the exchange. In the basic protocol, we thus have something weaker than a signature, but stronger than symmetric authentication. This is an interesting fact in itself; an obvious direct application is witnessed communication, where (for example) a videoconference is also viewed by a third party who may be called on to testify about some aspects of it later. Another is in communication systems with third party logging, such as the SWIFT system mentioned above.

However, our scheme can be converted quite simply into one with off-line nonrepudiation. The trick is a convention that each principal keeps secret their last password and reveals it to a judge in the event of a dispute. Alternatively, each principal could have a notary sign a hash of his last password together with a transcript of the session.

6 Other Practical Applications

We now consider a number of other applications of our technique.

6.1 An Integrity Equivalent of Diffie-Hellman

Our protocol allows us to link a number of incidents securely, and so we ask whether it has any particularly interesting uses for the identification of principals in computer networks. After all, a principal is in some sense just the linkage of a series of incidents.

In the real world, it is often only necessary to remember a certain distance back in such a chain. We may be quite unable to remember what incident first convinced us of the identity of our mother, or of many of our other relatives and friends; but the absence of a definitive initial authentication instance is considered to be irrelevant in such circumstances. Similar considerations may apply to electronic personae. Many people nowadays have built up relationships and even scientific collaborations over the net with other people whom they only later meet in person.

The above protocol for bidirectional authentication shows how we can interlock hash-based authentication by two different individuals at the same time. One novel implication of this is that two principals who do not originally share any secret can protect the serialisation of the traffic between. Once this integrity channel is established, it will guarantee both the content and the correct serialisation of all future messages.

This channel does for integrity what the Diffie Hellman protocol does for secrecy. This may seem counterintuitive, and it certainly challenges the common understanding that the *'man-in-the-middle attack can defeat any protocol not involving a secret'* [17]. What is actually happening of course is that a middleperson attacking the integrity channel has to participate in it from the start; she cannot join in later, or the views that the two participants of the transaction history will differ in nontrivial ways.

At the systems level, this is because we have set up a channel with integrity but no authenticity, in the sense that we do not know who we are speaking to. So Alice, who wanted to speak to Bob, might in fact be speaking to Charlie. However in this case Charlie will have to participate actively in the conversation between them for the rest of time if he wishes to escape detection; he will not be able to drop in and out of their traffic at will.

There are applications in which conventional authentication may not be possible and yet the limitation on active attacks that this technique provides might be valuable. An example is communication between dissidents in an oppressive country that compels the escrow of even signing keys. In such conditions, trust is likely to be built up slowly over a long series of messages, and users may well wish to be sure that a channel that they are starting to trust is not taken over by authority.

A curious feature of multiparty Guy Fawkes, however, is that when one principal introduces two others with whom he has established sessions, he cannot ever persuade either of them that the other actually exists. Alice, on being introduced to Bob by Charlie, could just as easily be introduced to another persona of the principal behind Charlie. This appears to be a feature of electronic communications in general; it is merely brought out when we start to consider protocols for establishing trust that do not rely on some bootstrapping event in the physical world. (The Rivest-Shamir interlock protocol is the only one we know of that can achieve a similar effect [24]; but previous comment on it has focussed on the understandable difficulty of using it for authentication [25].)

6.2 Tamper-evident audit trails

It is a well known problem that an intruder can often acquire root status by using well known operating system weaknesses, and then alter the audit and log information to remove the evidence of the intrusion. In order to prevent this, some Unix systems require that operations on log and audit data other than reads and appends be carried out from the system console. Others do not, and it could be of value to arrange alternative tamper-evidence mechanisms.

A first idea might be to simply sign and timestamp the audit trail at regular intervals, but this is not sufficient as a root intruder will be able to obtain the private signing key and retrospectively forge audit records. In addition, the intervals would have to be small (of the order of a second, or even less) and the computation of RSA or DSA signatures at this frequency could impose a noticeable system overhead.

In this application, the Guy Fawkes protocol appears well suited because of the low computational overhead (two hash function computations per signature) and the fact that all secrets are transient; this second's secret codeword is no use in forging a signature of a second ago.

The envisaged architecture here is that each server or other sensitive machine on a LAN would authenticate its log and audit data once per second (or even more frequently) with a local timestamping service, that would run on a machine stripped of vulnerabilities such as sendmail. This could in its turn interact at some suitable frequency with an external machine such as a corporate time stamping service, which in turn could interact with a commercial service. The protocols for this are currently under development.

6.3 Secure access to timestamping services

Third party timestamping services have much wider uses than simply providing trust backup for audit data; they are used to provide evidence of priority for all kinds of intellectual property, financial records and other business documents. An example design of such a service is found in that of Haber and Stornetta [23]. There the messages to be stamped are hashed in a tree, with a hash of

all input messages being made available once a second over the web and once a week through a newspaper advertisement. A signer can incorporate the relevant part of this hash tree with the disclosure of his message in the same way that he would incorporate a collection of certificates.

This immediately raises the question of how the user can trust that the timestamping service that appears to have incorporated her message into its tree is a genuine one, and not a simulacrum created by an attacker who has taken over her network connection. A conventional approach would be for the timestamping service to affix a digital signature to the timestamps it returns. However, this brings extra complexity into the trust loop, with possible attendant costs of licensing a digital signature technology. There are also performance issues in signature generation when providing a service sized to generate a thousand timestamps a second, as Haber and Stornetta's system is; this can be provided on a workstation which will however only generate 50 RSA signatures per second.

The simple solution is to use the Guy Fawkes protocol as a means of authenticating the timestamping service to the user. We are currently working on an implementation that will work with this timestamping service.

6.4 Other applications

Other specific applications in which the Guy Fawkes protocol might offer advantages over other integrity and nonrepudiation mechanisms include the updating of root keys used by software vendors and CAs; telegambling; digital elections; membership of clubs with optional anonymity; and software metering mechanisms in which a vendor sends 'keep-alive' messages to the systems of those subscribers who keep on paying their licence fees. The value that Guy Fawkes can provide here lies in the absence of a single short long term secret that a pirate could broadcast.

Another family of applications is in general authentication and non-repudiation protocols where for cost reasons it is desired to use low-power processors, such as cheap smartcards or microcontrollers. In general, where we have an interactive application in which some combination of anonymity with either serialisation or temporary nonrepudiation is required, a protocol based on Guy Fawkes may be the tool for the job.

Finally, given the politics of cryptography, it may be worth remarking that all secrets in the Guy Fawkes protocol become known; there are no long-term user secrets that can be used as decryption keys and thus less motive to attempt to escrow signing keys, with the consequential loss of evidential reliability. It can of course be used to detect middleperson attacks on Diffie Hellman key exchange, and thus to set up confidential channels indirectly. However it is unclear that any nonrepudiation — or even authentication — can be achieved if preventing authentic Diffie-Hellman is a national policy imperative; even a simple password is enough to prevent middleperson attacks [22].

7 Conclusion

We have shown that it is possible to provide a non-repudiation service without public key mechanisms, tamper resistance or third party logging. We do not even require any principal to possess a long-term secret.

This involves a new protection primitive which in its simplest form behaves extremely like a digital signature and may be obtained at a negligible computational cost, provided that a timestamp service exists. We have also shown a bidirectional primitive that may be used to authenticate digital streams at much less cost than the previous best protocol. This led us to an 'integrity equivalent' of Diffie Hellman: two users can under quite reasonable assumptions establish a channel whose traffic is protected against modification, without either of them possessing a secret at the start of the protocol or concealing any secrets from authority.

Quite apart from possible applications, these constructions raise a number of interesting questions, such as: what exactly is a digital signature? what is the necessary role of communication in a public key infrastructure? and what tradeoffs are there between computation, communication and the maintenance of state?

References

1. "The History of Subliminal Channels", GJ Simmons, in *Proceedings of the First International Workshop on Information Hiding* (Springer LNCS v 1174) pp 237-256
2. "Verification of Treaty Compliance — Revisited", GJ Simmons, in *Proceedings of the IEEE Symposium on Security and Privacy* (IEEE, 1983) pp 61-66
3. "Constructing digital signatures from a one-way function", L Lamport, *SRI TR CSL 98* (1979)
4. "A Digital Signature Based on a Conventional Encryption Function" RC Merkle, in *Advances in Cryptology — Crypto 87* (Springer LNCS v 293) pp 369-378
5. "A Certified Digital Signature", RC Merkle, in *Advances in Cryptology — Crypto 89* (Springer LNCS v 435) pp 218-238
6. "On-line / off-line digital signatures", S Even, O Goldreich, S Micali, in *Advances in Cryptology — Crypto 89* (Springer LNCS v 435) pp 263-275
7. "Directed Acyclic Graphs, One-way Functions and Digital Signatures", D Bleichenbacher, UM Maurer, *Advances in Cryptology — Crypto 94* (Springer LNCS v 839) pp 75-82
8. "The S/KEY One-Time Password System", N Haller, in *Proceedings of the ISOC Symposium on Network and Distributed System Security* (February 1994, San Diego, CA) pp 151 - 157; see also RFCs 1704, 1760 and 1938
9. "NetCard — A Practical Electronic Cash System", R Anderson, C Manifavas, C Sutherland, in *Proceedings of the Fourth Cambridge Security Protocols Workshop* (Springer LNCS v 1189) pp 49-57
10. "PayWord and MicroMint: Two Simple Micropayment Schemes", RL Rivest, A Shamir, in *Proceedings of the Fourth Cambridge Security Protocols Workshop* (Springer LNCS v 1189) 69-87

11. "Electronic Payments of Small Amounts", TP Pedersen, in *Proceedings of the Fourth Cambridge Security Protocols Workshop* (Springer LNCS v 1189) 59-68
12. "New Directions in Cryptography", W Diffie, ME Hellman, in *IEEE Transactions on Information Theory* v IT-22 no 6 (November 1976) pp 644-654
13. "The First Ten Years of Public-Key Cryptography", W Diffie, in *Proceedings of the IEEE* v 76 no 5 (May 88) pp 560-577
14. "How To Prove Yourself: Practical Solutions to Identification and Signature problems", A Fiat, A Shamir, in *Advances in Cryptology — CRYPTO 86*, Springer LNCS v 263 pp 186-194
15. "A Digital Signature Scheme Secure Against Adaptive Chosen Message Attacks", S Goldwasser, S Micali, RL Rivest, in *SIAM Journal of Computing* v 17 no 2 (April 1988) pp 281-308
16. "Digital Signatures with Blindfold Arbitrators who Cannot Form Alliances", SG Akl, in *Proceedings of the 1983 IEEE Computer Society Symposium on Security and Privacy*, pp 129-135
17. *'Applied Cryptography'*, B Schneier, Wiley 96
18. "Universal One-Way Hash Functions and Their Cryptographic Application", M Naor, M Yung, in *Proceedings of the 21st Annual ACM Symposium on the Theory of Computing* (1989) pp 33-43
19. "One-Way Functions are Necessary and Sufficient for Digital Signatures", J Rompel, in *Proceedings of the 22nd Annual ACM Symposium on the Theory of Computing* (1990) pp 387-394
20. *'Digital Signature Schemes — General Framework and Fail-Stop Signatures'*, B Pfitzmann, Springer LNCS v 1100
21. "How to Sign Digital Streams", R Gennaro, P Rohatgi, in *Advances in Cryptology — CRYPTO 97*, Springer LNCS v 1294 pp 180-197
22. "On fortifying key negotiation schemes with poorly chosen passwords", RJ Anderson, TMA Lomas, in *Electronics letters* v 30 no 12 (23rd July 1994) pp 1040-1041
23. "How to Time-Stamp a Digital Document", S Haber, WS Stornetta, in *Journal of Cryptology* v 3 no 2 (1991) pp 99-112
24. "How to Expose an Eavesdropper", RL Rivest, A Shamir, in *Communications of the ACM* v 27 no 4 (Apr 84) pp 393-395
25. "An Attack on the Interlock protocol When Used for Authentication", SM Bellovin, M Merritt, *IEEE Transactions on Information Theory* v 40 no 1 (Jan 94) pp 273-275
26. "Secure Agreement Protocols: Reliable and Atomic Group Multicast in Rampart", MK Reiter, in *Proceedings of the 1994 ACM Conference on Computer and Communications Security* pp 68-80
27. "Maintaining Security in the Presence of Transient Faults", R Canetti, A Herzberg, in *Advances in Cryptology — CRYPTO 94*, Springer LNCS v 839 pp 425-438
28. "Network Randomization Protocol: A Proactive Pseudo-Random Generator", CS Chow, A Herzberg, in *Usenix Security 95* pp 55-63
29. "The Omega Key Management Service", MK Reiter, MK Franklin, JB Lacy, RA Wright, in *Proceedings of the 1996 ACM Conference on Computer and Communications Security* pp 38-47